

PENDOLO A TORSIONE

FRANCESCO FORCHER

Università di Padova, Facoltà di Fisica
francesco.forcher@studenti.unipd.it
Matricola 1073458

DAVIDE CHIAPPARA

Università di Padova, Facoltà di Fisica
davide.chiappara@studenti.unipd.it
Matricola 1070160

SIMONE FRAU

Università di Padova, Facoltà di Fisica
simone.frau@studenti.unipd.it
Matricola 1074028

10 giugno 2014

$$pV = nRT$$

Sommario

Obiettivo dell'esperienza è stato studiare il moto di un cilindro in acqua in seguito ad una forza periodica impressa al cilindro stesso da un motore elettrico attraverso un filo che permette la trasmissione di un momento torcente. Dai campioni ricavati sperimentalmente si è indagato, in particolare, sulla curva di risonanza, sulla frequenza di risonanza, sul coefficiente di smorzamento e sulla pulsazione propria del sistema.

INDICE

I. APPARATO STRUMENTALE

L'apparato strumentale consiste in un cilindro in plexiglass al cui interno è posto un peso in acciaio di massa: $(115.5 \pm 0.1)g$ e diametro: $(22.7 \pm 0.1)mm$ e altezza: $(34.0 \pm 0.1)mm$ collegato ad un filo di acciaio armonico, materiale dotato discrete capacità elastiche, ed immerso in un liquido. Il filo è inoltre collegato ad una piattaforma rotante azionata da un motore di diametro circa 8cm che, una volta azionato, induce un'oscillazione forzata del corpo formato dal filo più il pesetto. Il range delle frequenze alla quale il motore può essere indotto ad oscillare risulta compreso tra 0.8 e 1.2 Hz. Di suddetta oscillazione è possibile modificare il periodo e l'ampiezza. Il tutto viene controllato e registrato mediante l'interfaccia fornita da un computer. I dati vengono acquisiti in intervalli di 0.05 secondi permettendo una frequenza di rilevamento di 20 dati per secondo. L'interfaccia permette di visualizzare valori della frequenza, dell'ampiezza e della fase (espressa in gradi). Inoltre sono presenti diversi grafici che mostrano, in tempo reale, la curva di risonanza, il moto del pesetto, segnandone la posizione. Inoltre la presenza del pulsante offset permetteva di tarare l'apparato dopo ogni misurazione, al fine di limitare errori sistematici.

II. METODOLOGIA DI MISURA

Per poter stimare la frequenza di risonanza si è proceduti azionando il motore e mettendo in oscillazione la piattaforma rotante. Partendo dalle informazioni fornite e dall'apparecchiatura si è deciso di porre l'ampiezza a 10 millesimi di giro e di variare gli intervalli delle frequenze di 0.020 Hz e acquisendo campioni di misure per un tempo totale di circa 10 secondi durante la fase a regime, interrompendo la misurazione per eseguire lo store dei dati così ricavati, spegnendo il motore e intraprendendo una seconda fase di registrazione dati per una durata di circa 20 secondi per la fase di smorzamento, al termine del quale è stato eseguito nuovamente lo store dei dati. Le misure sono state prese solo non appena è stato evidente dai grafici di riferimento il carattere periodico del moto del pendolo. L'apparato strumentale è stato ricalibrato prima delle prese dati della giornata per permetterne un funzionamento ottimale. L'ampiezza massima di oscillazione della forzante è stata scelta per permettere oscillazioni abbastanza ampie da studiare ma non così ampie da rendere caotico il moto del pendolo, costringendolo a muoversi sul piano perpendicolare all'asse di oscillazione. Attraverso questo metodo è stato possibile ottenere una panoramica del comportamento oscillatorio del materiale di studio e identificare efficacemente il settore in cui avveniva il fenomeno di risonanza. Tale settore è stato poi sondato ricorrendo al metodo di bisezione restringendosi in un intorno di valori della frequenza e aumentando l'esposizione dell'acquisizione dati. In questa fase (che si concentra nell'intervallo tra 0.965 e 0.970 Hz) sono stati registrati valori per la durata di circa 100 secondi per la fase a regime e di 40 secondi circa per la fase in smorzamento, una volta spento il motore. Questo offre agli sperimentatori la possibilità di analizzare una serie di campioni più concentrati avente intervalli di frequenza di 0.001 Hz, permettendo di stimare il più efficacemente possibile la frequenza di risonanza.

III. PRESENTAZIONE DEI DATI

I. Tabelle

Tabella 1: *Pulsazioni smorzate*

Frequenza forzante [Hz]	Pulsazione smorzante [$\frac{rad}{s}$]	Errore [Hz]
0.900	6	1
0.920	6.1	0.2
0.940	6.1	0.2
0.960	6.08	0.03
0.965	6.1	0.1
0.966	6.07	0.06
0.967	6.08	0.06
0.968	6	1
0.969	5	2
0.970	6.08	0.08
0.975	6.0	0.6
0.980	6.08	0.06
0.990	6.1	0.1
1.000	6	1
1.020	6.1	0.5
1.060	5	2
1.080	6	1

Media pesata pulsazione di smorzamento: $(6.07 \pm 0.09) [\frac{rad}{s}]$

Tabella 2: *Ampiezze di osillazione a regime*

Frequenza forzante [Hz]	Ampiezza [giri]	σ_{amp} [giri]
0.900	0.031	0.001
0.920	0.0452	0.0009
0.940	0.083	0.001
0.960	0.2187	0.0007
0.965	0.247	0.008
0.966	0.23	0.01
0.967	0.282	0.002
0.968	0.290	0.006
0.969	0.290	0.002
0.970	0.275	0.002
0.975	0.227	0.001
0.980	0.168	0.003
1.000	0.072	0.003
1.020	0.050	0.002
1.040	0.0373	0.0007
1.060	0.0302	0.0005
1.080	0.0257	0.0007
1.100	0.0226	0.0006

Tabella 3: Interpolazione per trovare le gamma, retta $y = a + b \cdot x$ su scala logaritmica

Frequenza forzante [Hz]	Parametro a	Errore su a	Parametro b [Hz]	Errore su b [Hz]
0.900	-1.56	0.02	-0.045	0.002
0.920	-1.22	0.04	-0.047	0.003
0.940	-0.82	0.05	-0.040	0.005
0.960	0.293	0.009	-0.0482	0.0008
0.965	0.459	0.006	-0.0481	0.0003
0.966	0.461	0.005	-0.0481	0.0002
0.967	0.51	0.02	-0.0470	0.0008
0.968	0.53	0.02	-0.0467	0.0004
0.969	0.52	0.03	-0.0469	0.0005
0.970	0.42	0.04	-0.045	0.002
0.975	0.23	0.02	-0.046	0.001
0.980	-0.14	0.06	-0.039	0.005
1.000	-0.85	0.09	-0.035	0.008
1.020	-1.25	0.02	-0.045	0.002
1.060	-2.05	0.02	-0.048	0.002
1.080	-2.2	0.1	-0.03	0.01

Media pesata gamma: $(-0.047 \pm 0.007)[Hz]$

Pulsazione propria: $(6.1 \pm 0.3)[Hz]$

II. Grafici

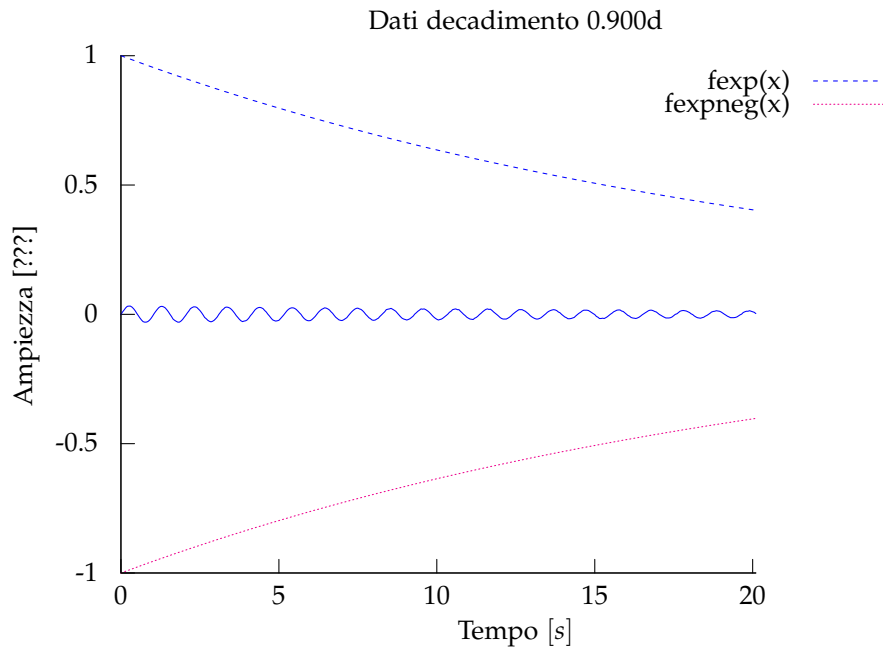
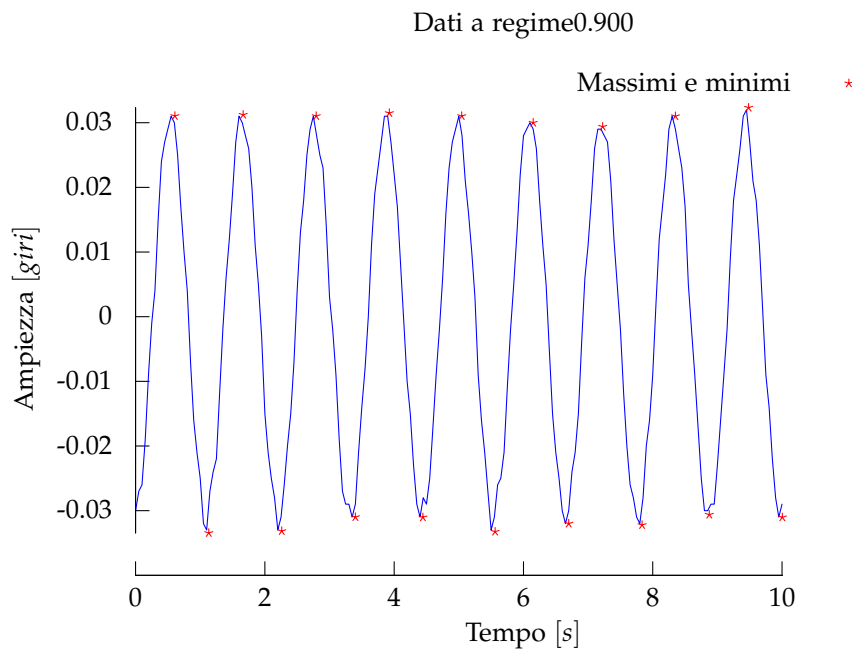
Grafico 1 Grafico 0.900dgdecad.tex**Grafico 2** Grafico 0.900greg.tex

Grafico 3 Grafico 0.920dgdecad.tex

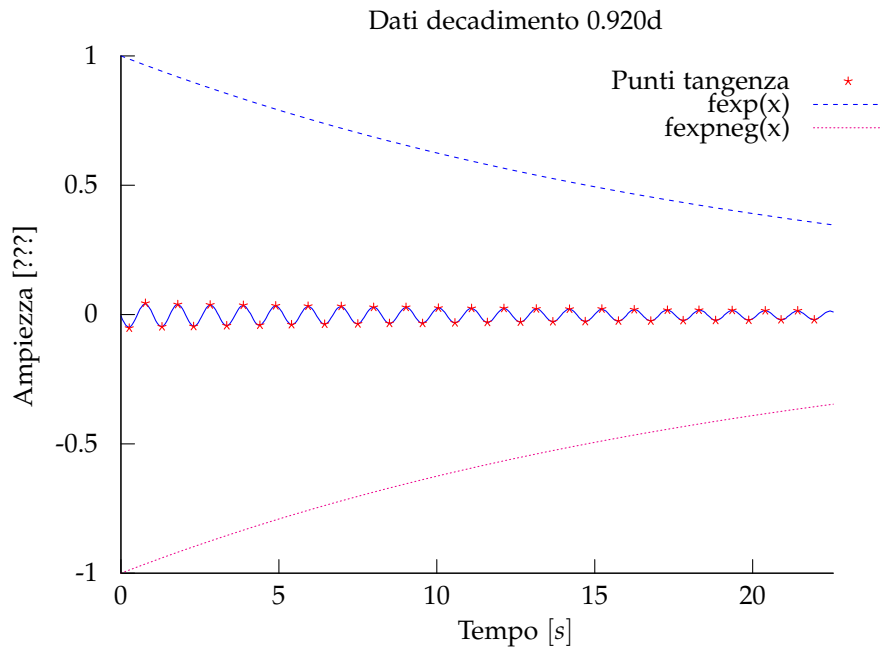


Grafico 4 Grafico 0.920greg.tex

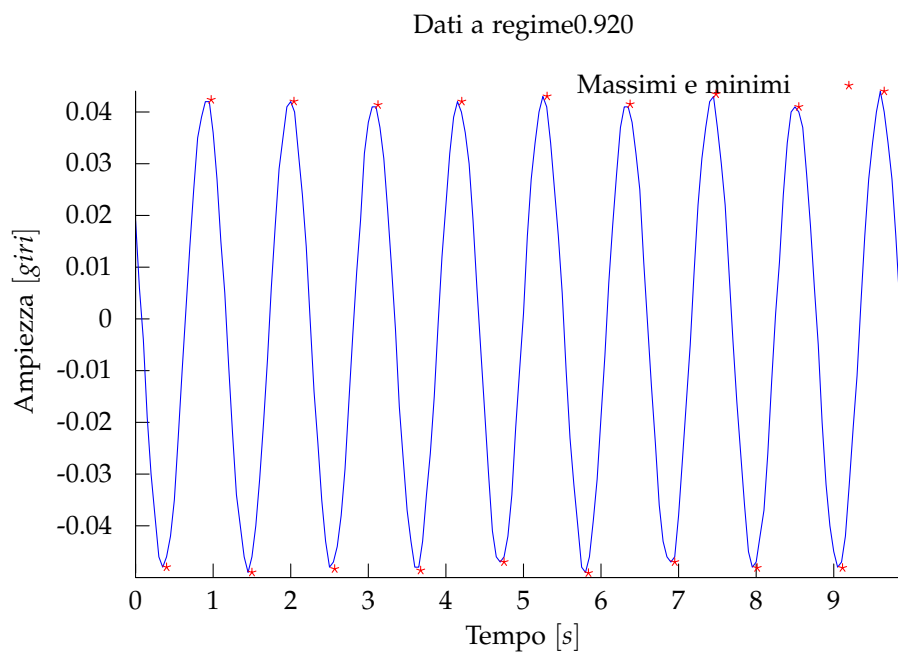


Grafico 5 Grafico 0.940dgdecad.tex

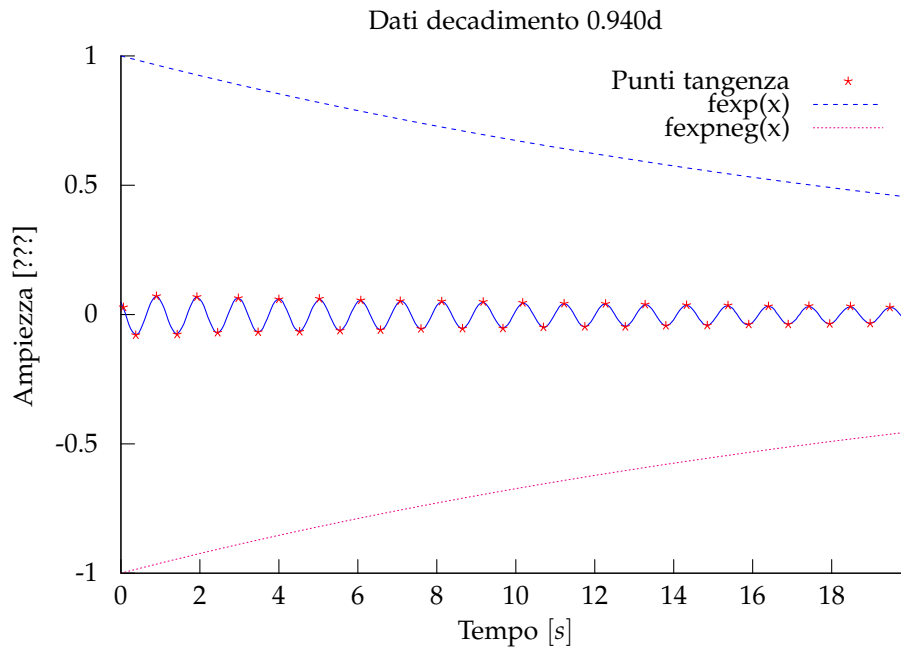


Grafico 6 Grafico 0.940greg.tex

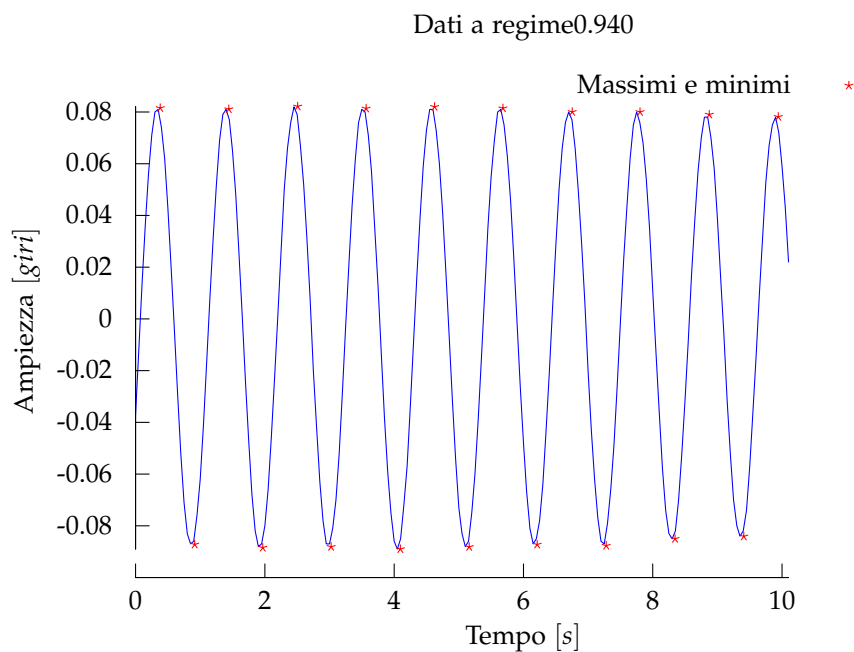


Grafico 7 Grafico 0.960dgdecad.tex

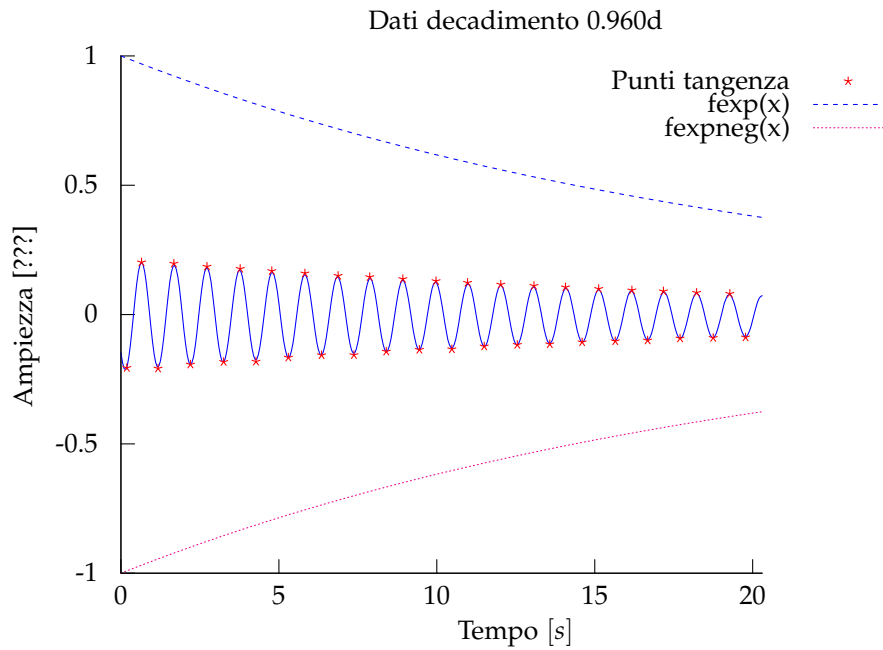


Grafico 8 Grafico 0.960greg.tex

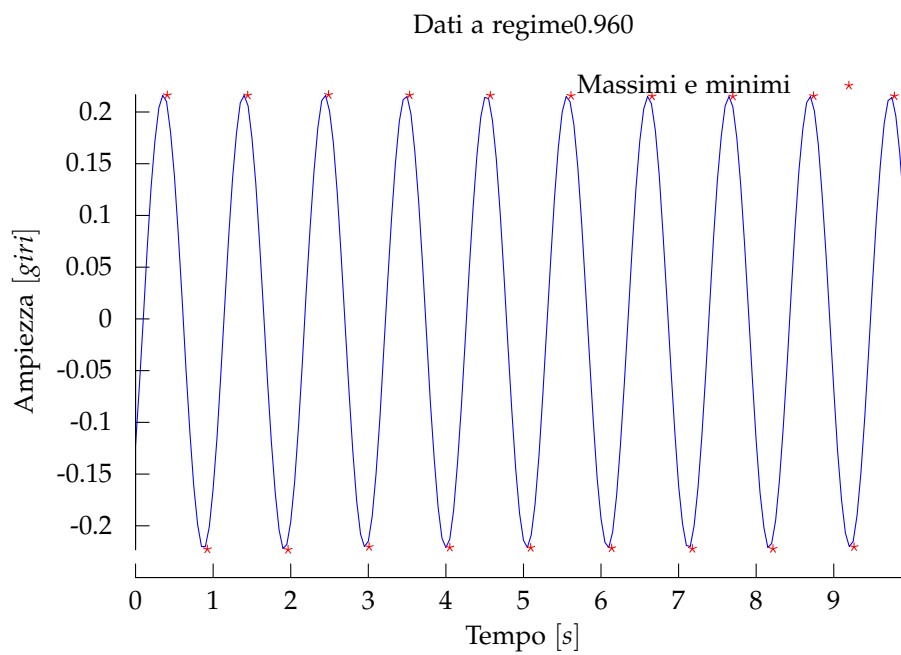


Grafico 9 Grafico 0.965dgdecad.tex

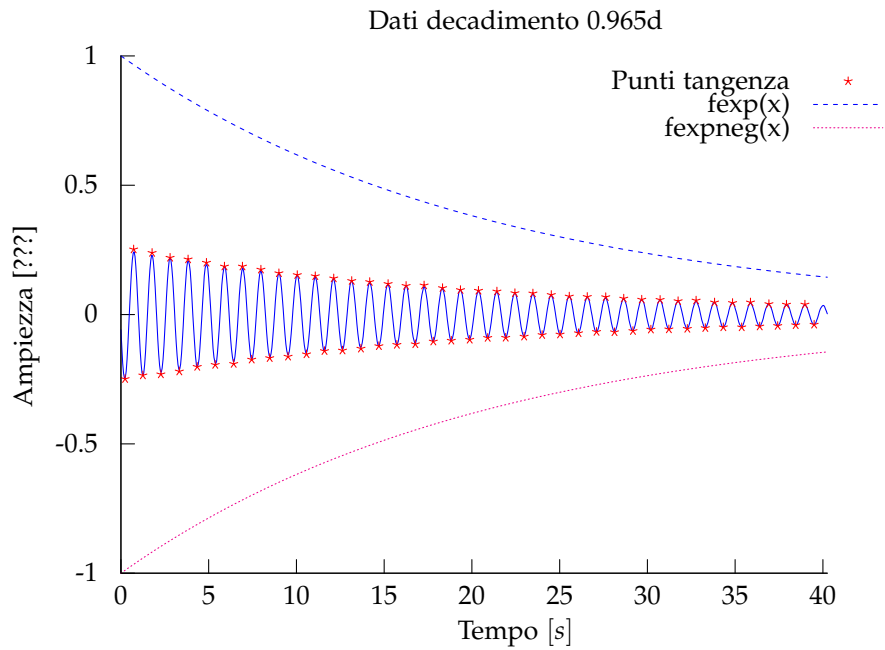


Grafico 10 Grafico 0.965greg.tex

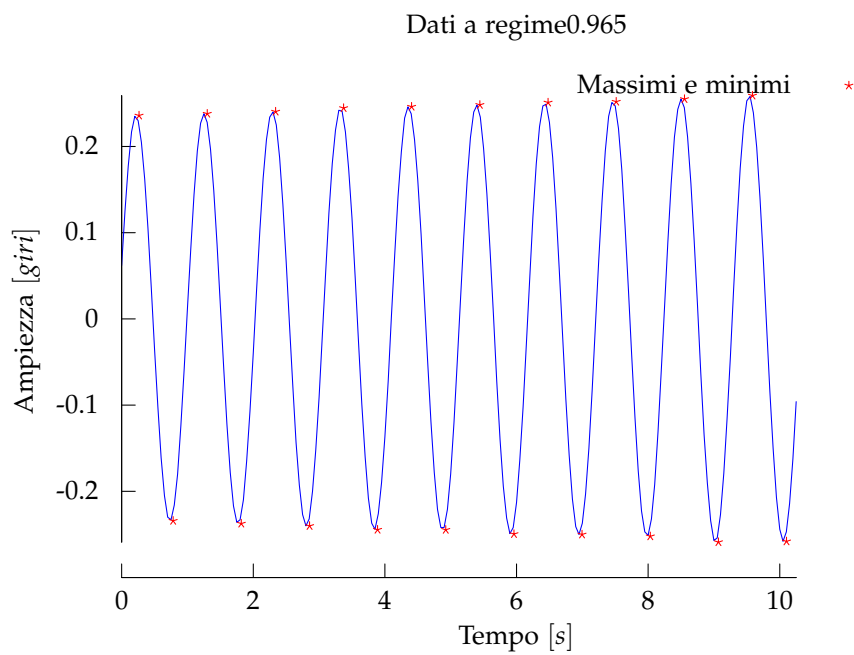


Grafico 11 Grafico 0.966dgdecad.tex

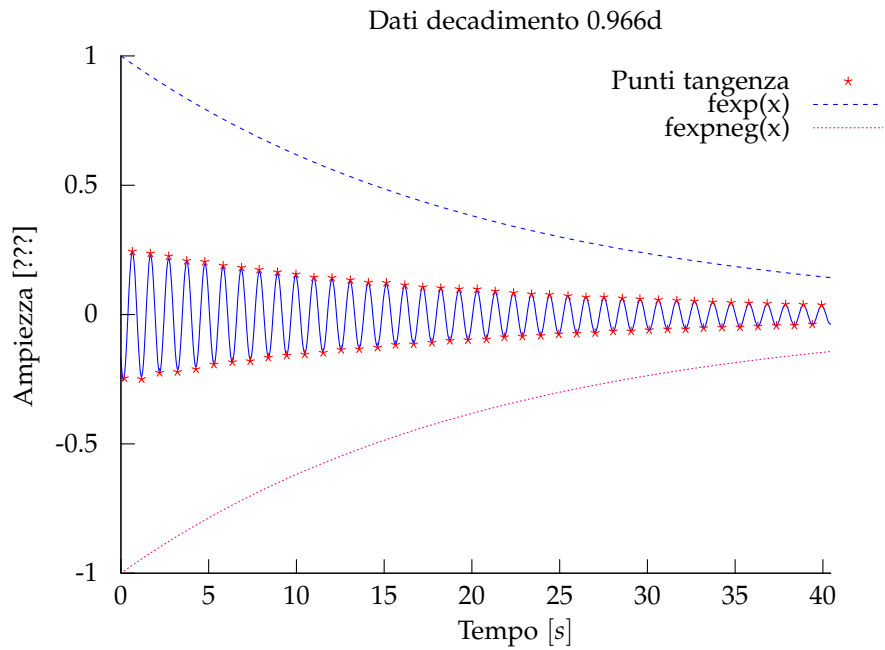


Grafico 12 Grafico 0.966greg.tex

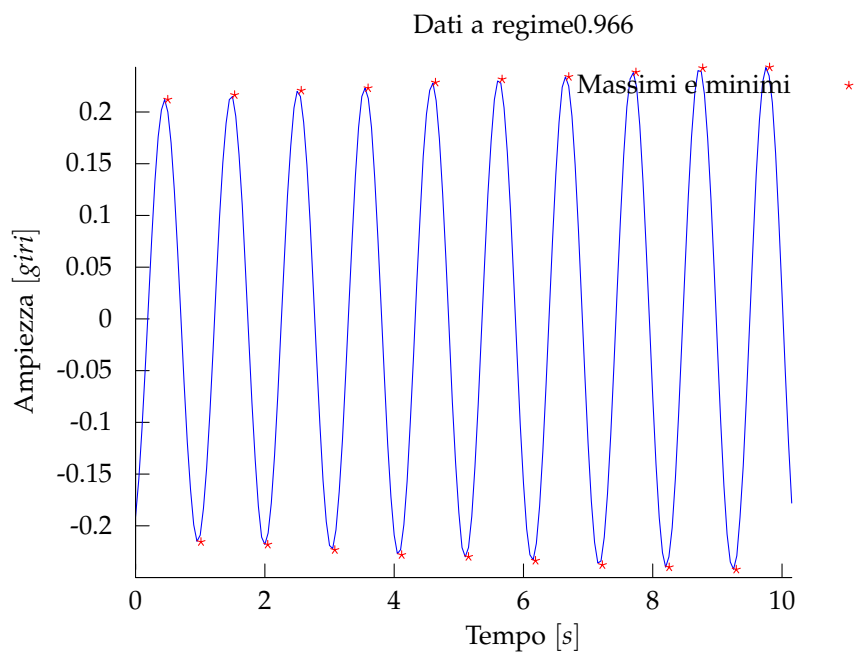


Grafico 13 Grafico 0.967dgdecad.tex

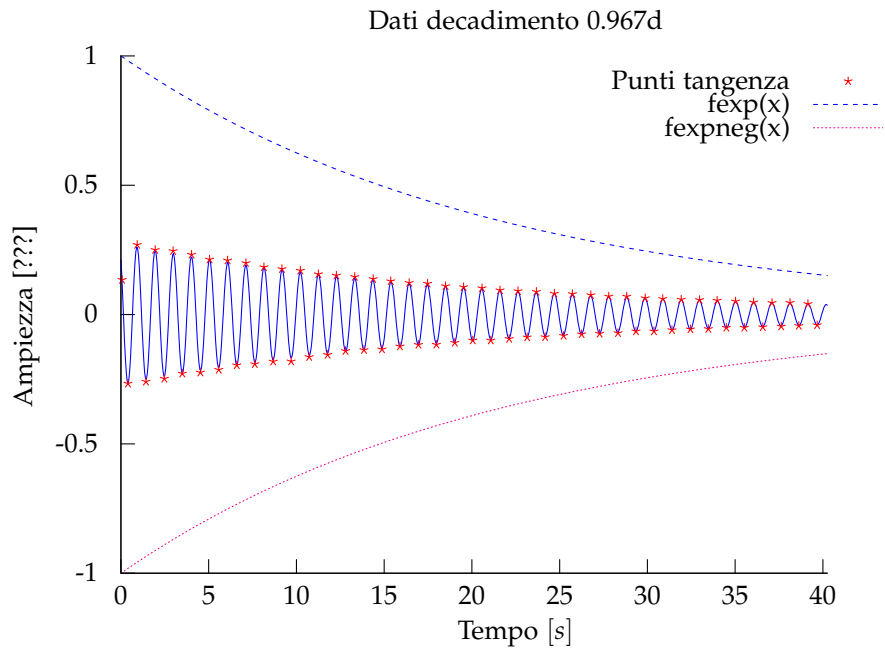


Grafico 14 Grafico 0.967greg.tex

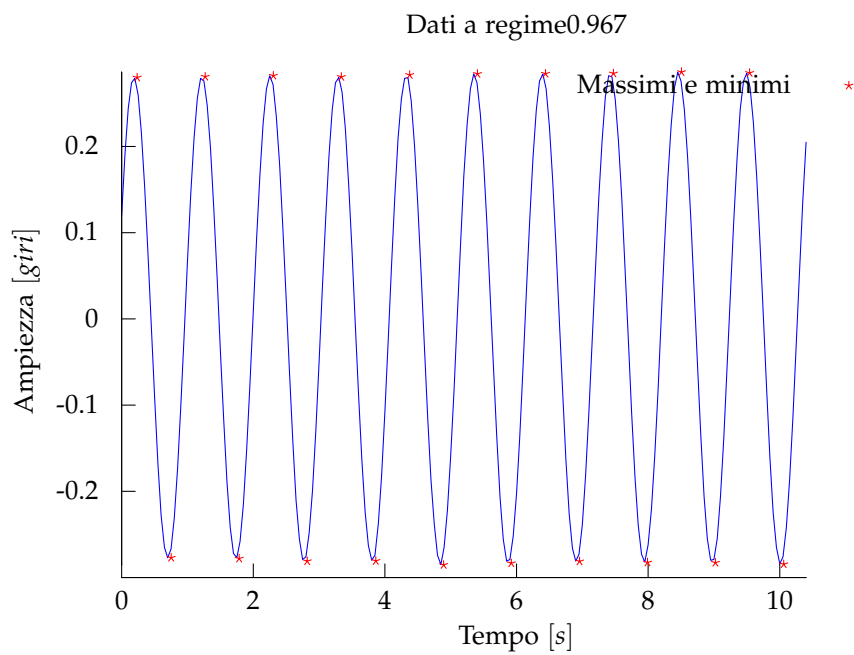


Grafico 15 Grafico 0.968dgdecad.tex

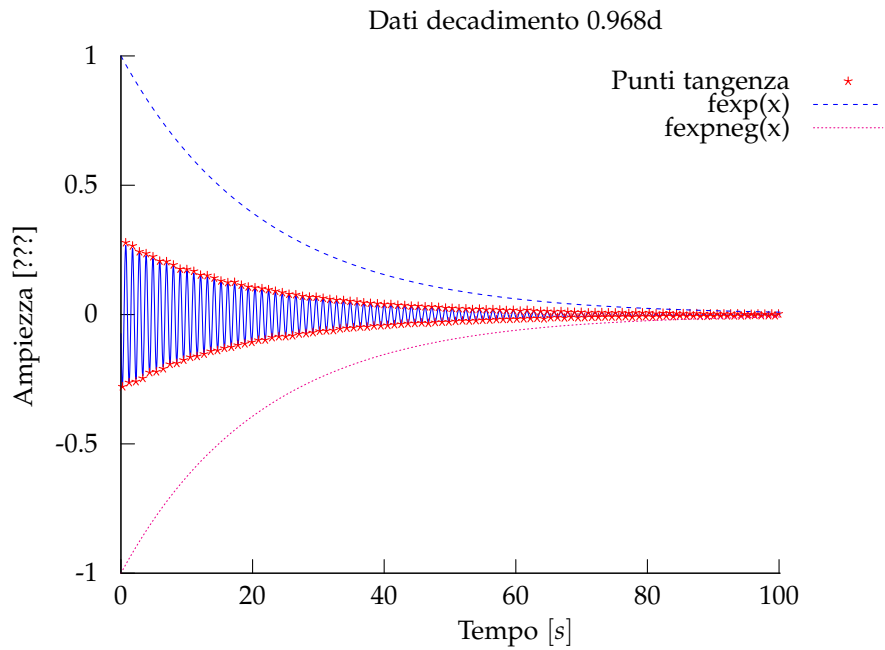


Grafico 16 Grafico 0.968greg.tex

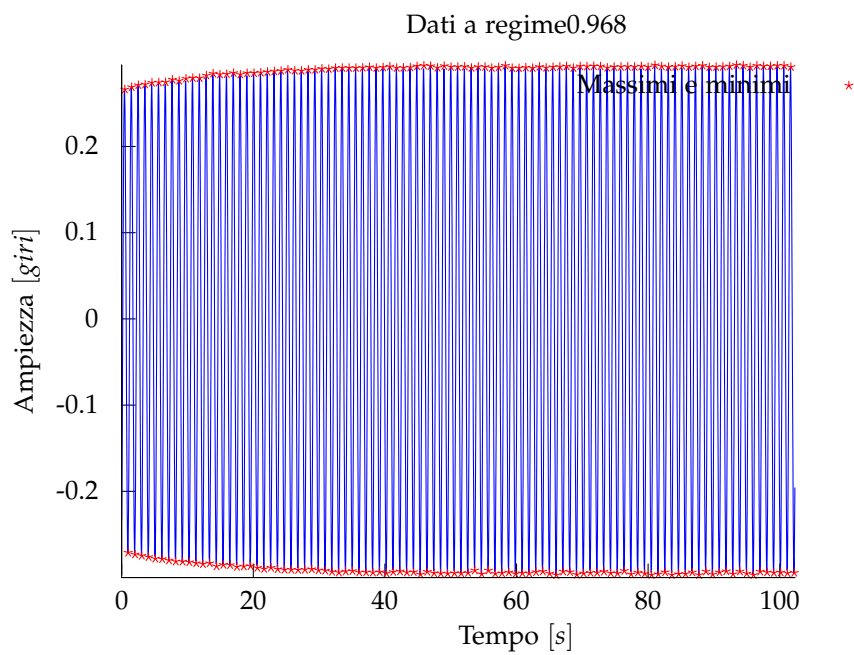


Grafico 17 Grafico 0.969dgdecad.tex

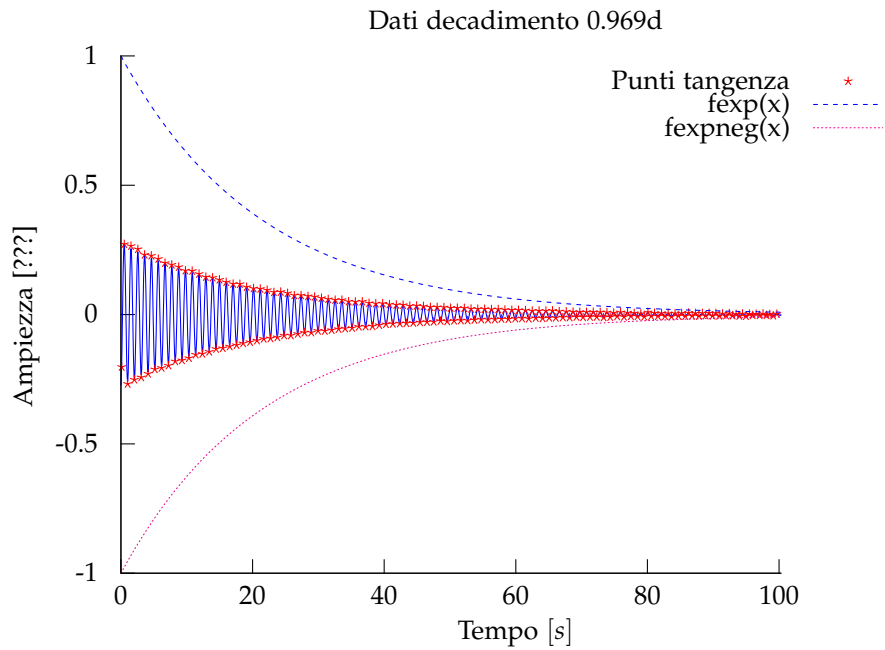


Grafico 18 Grafico 0.969greg.tex

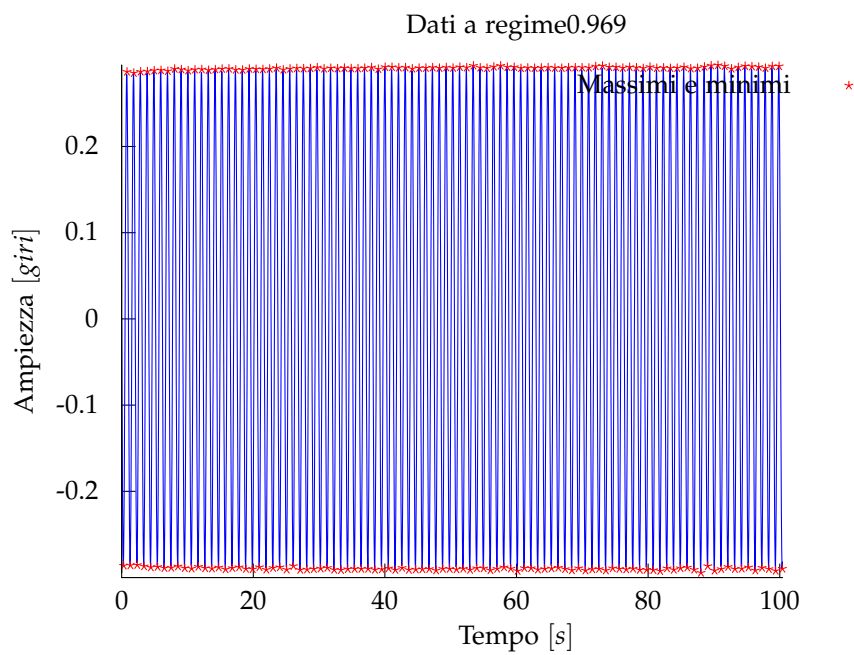


Grafico 19 Grafico 0.970dgdecad.tex

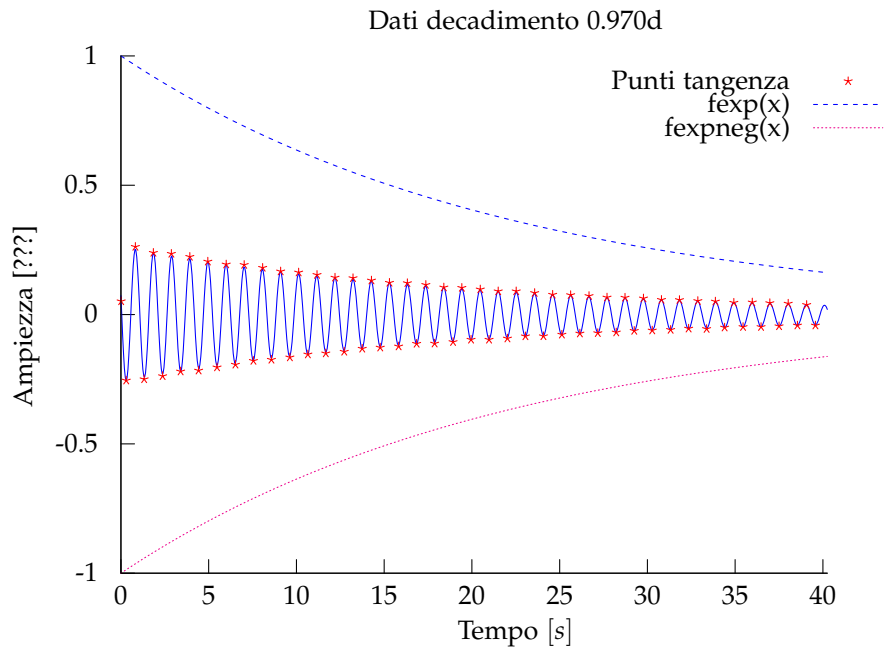


Grafico 20 Grafico 0.970greg.tex

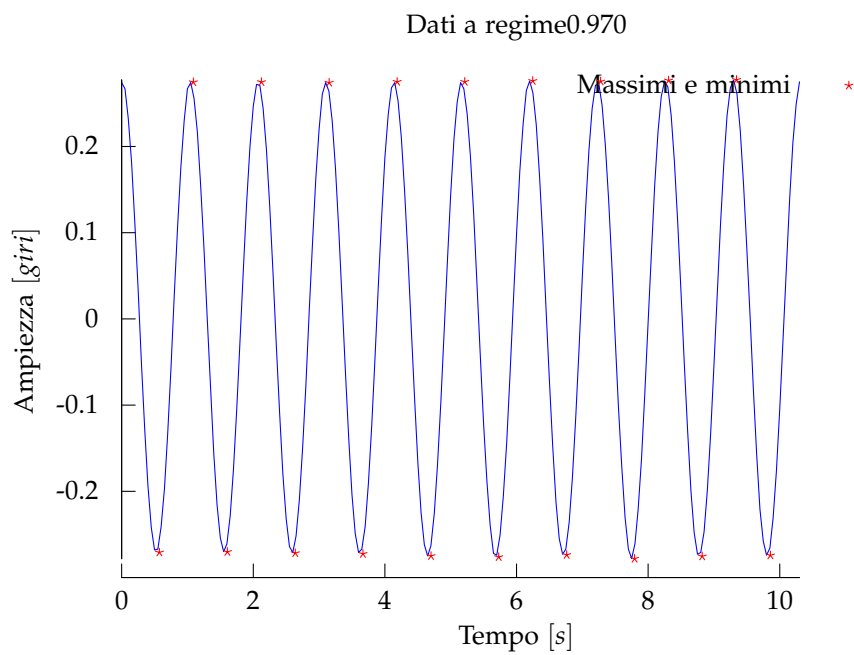


Grafico 21 Grafico 0.975dgdacad.tex

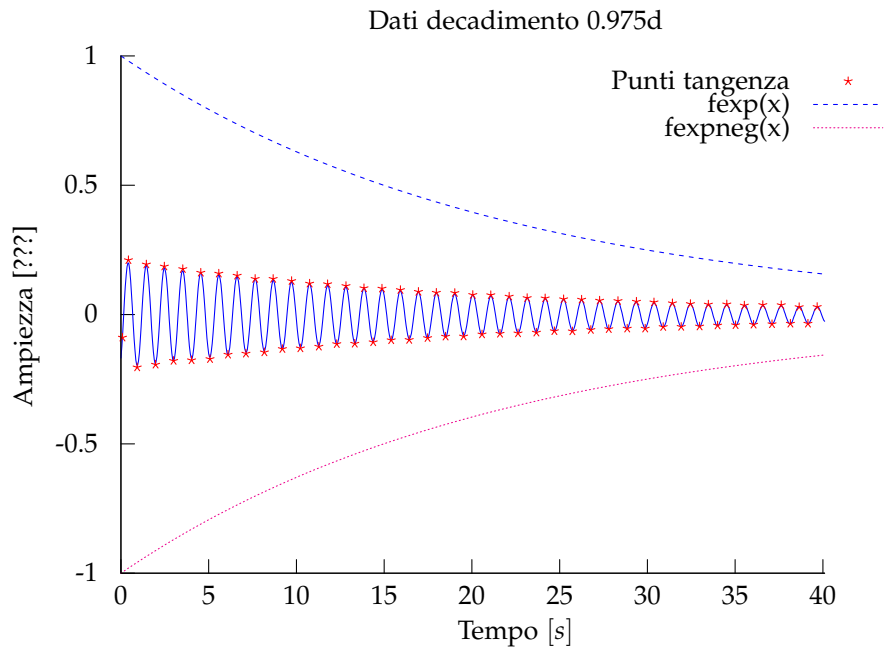


Grafico 22 Grafico 0.975greg.tex

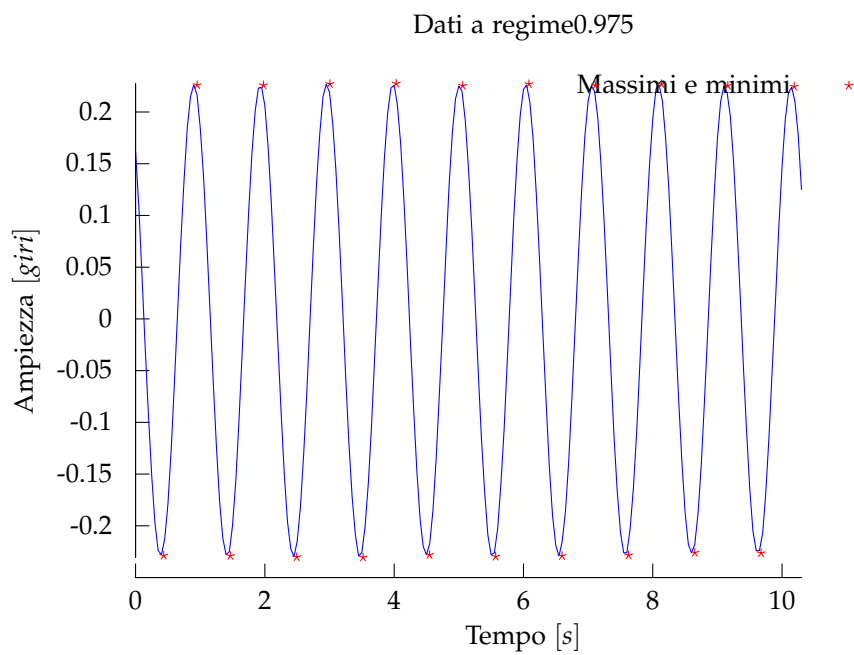


Grafico 23 Grafico 0.980dgdecad.tex

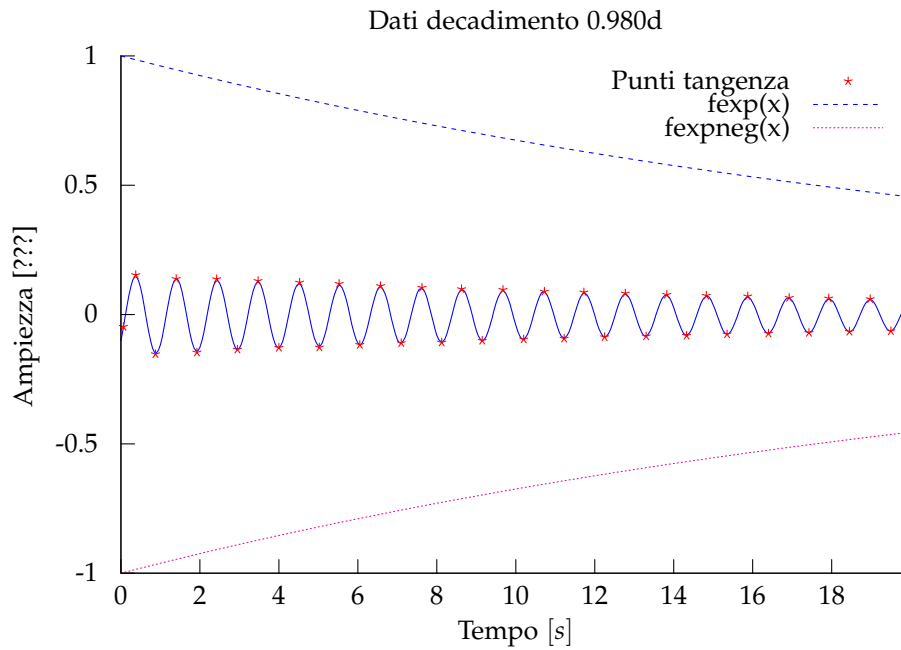


Grafico 24 Grafico 0.980greg.tex

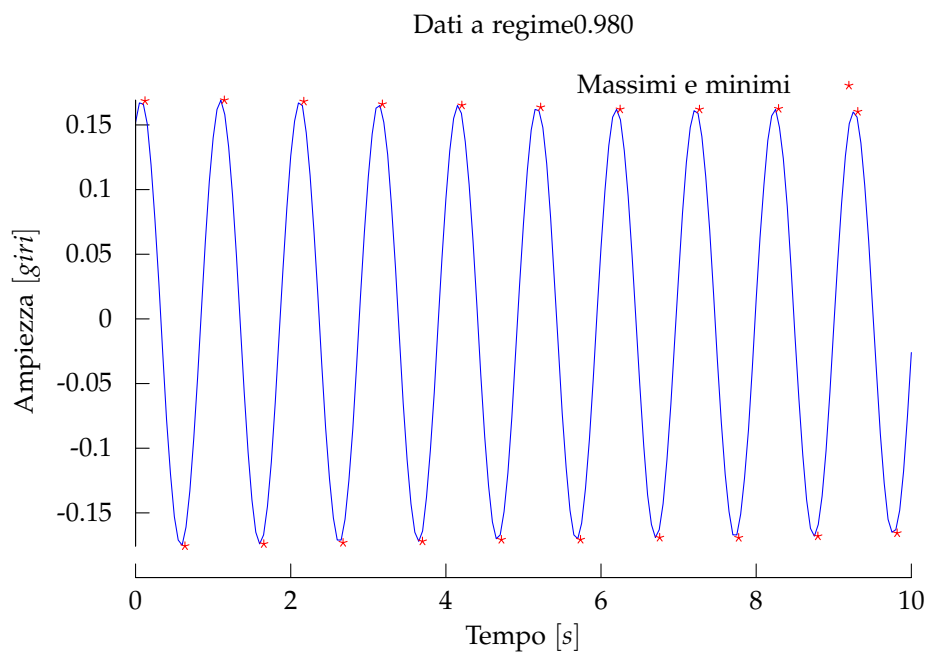


Grafico 25 Grafico 1.000dgdecad.tex

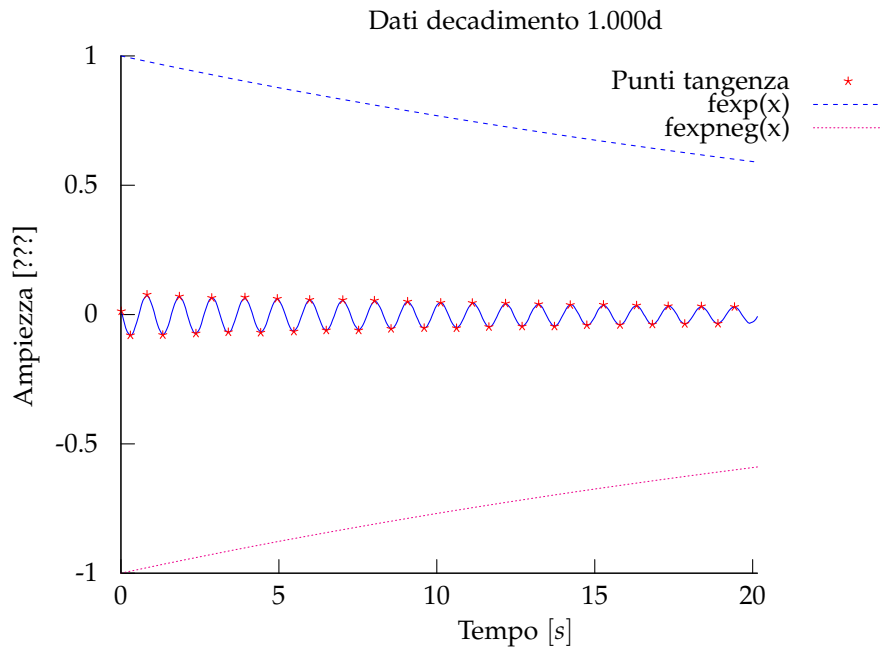


Grafico 26 Grafico 1.000greg.tex

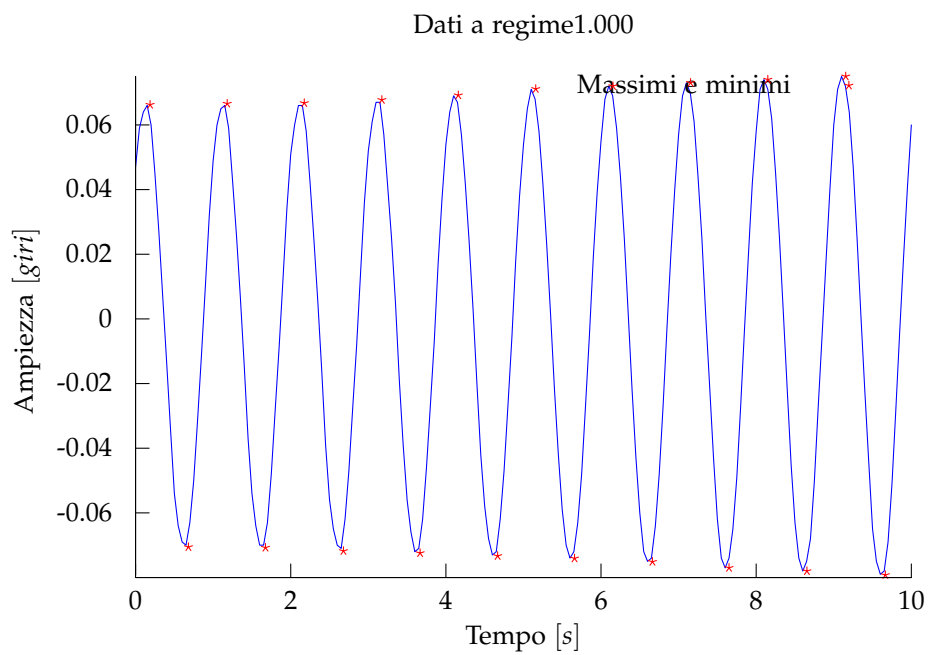


Grafico 27 Grafico 1.020dgdecad.tex

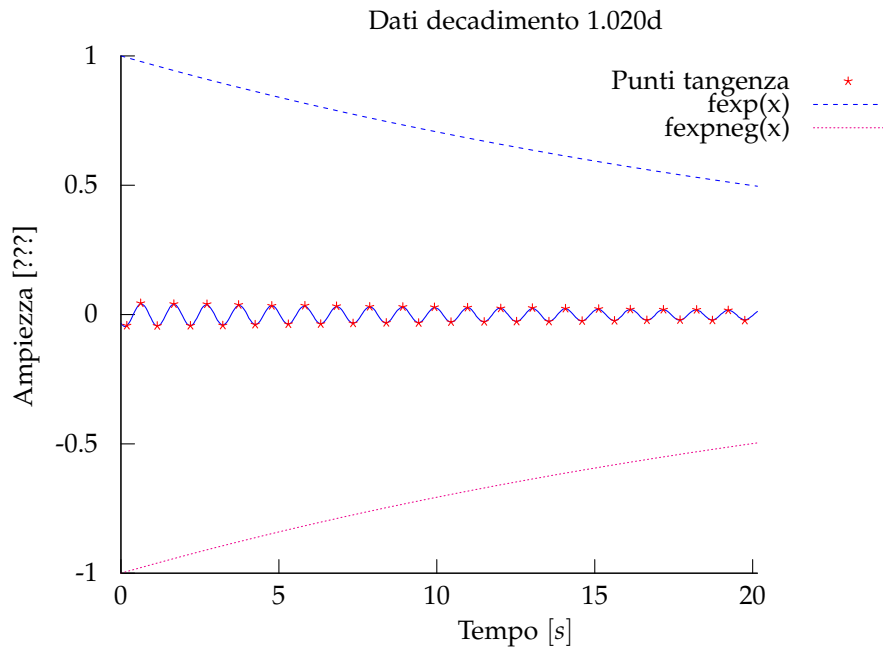


Grafico 28 Grafico 1.020greg.tex

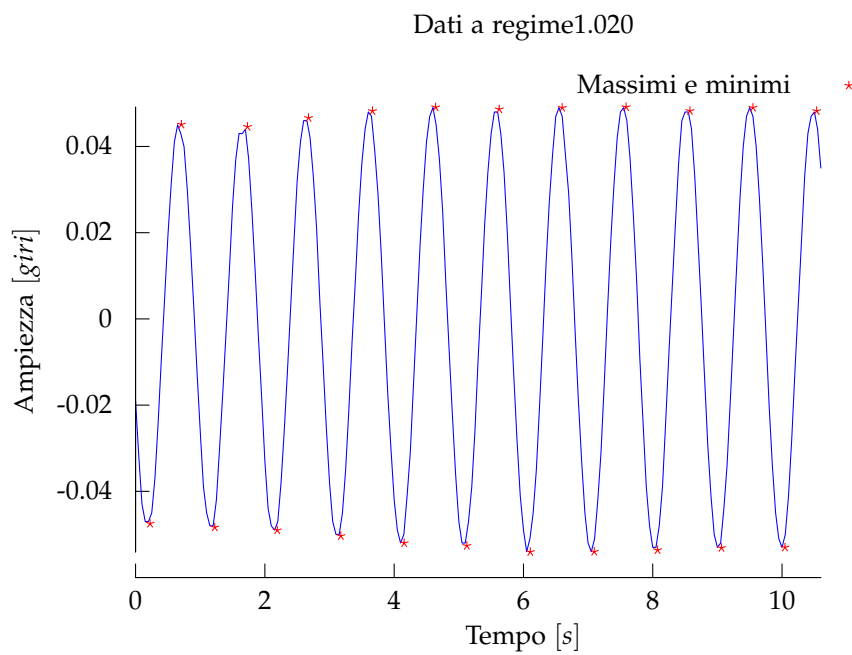


Grafico 29 Grafico 1.060dgdecad.tex

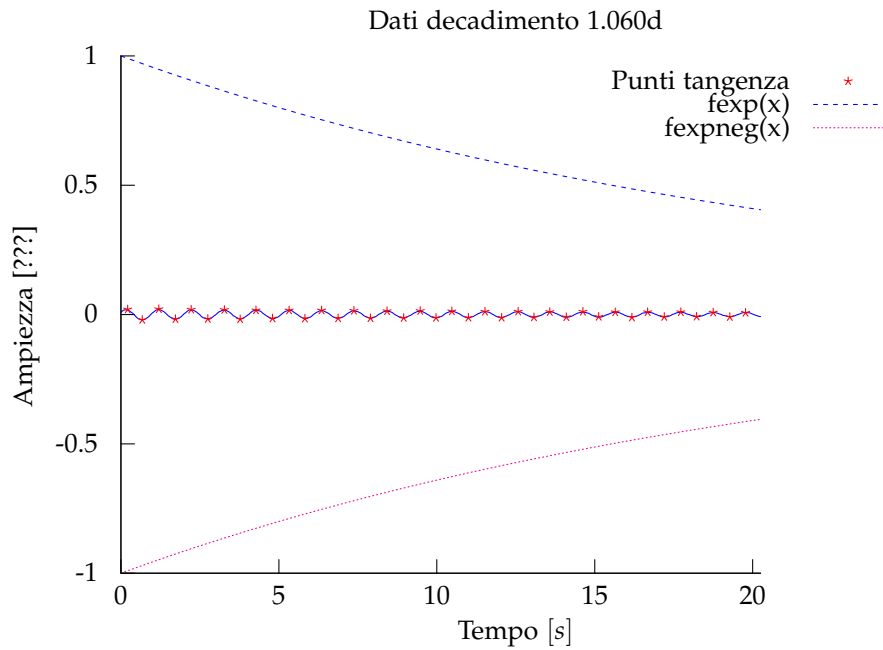


Grafico 30 Grafico 1.060greg.tex

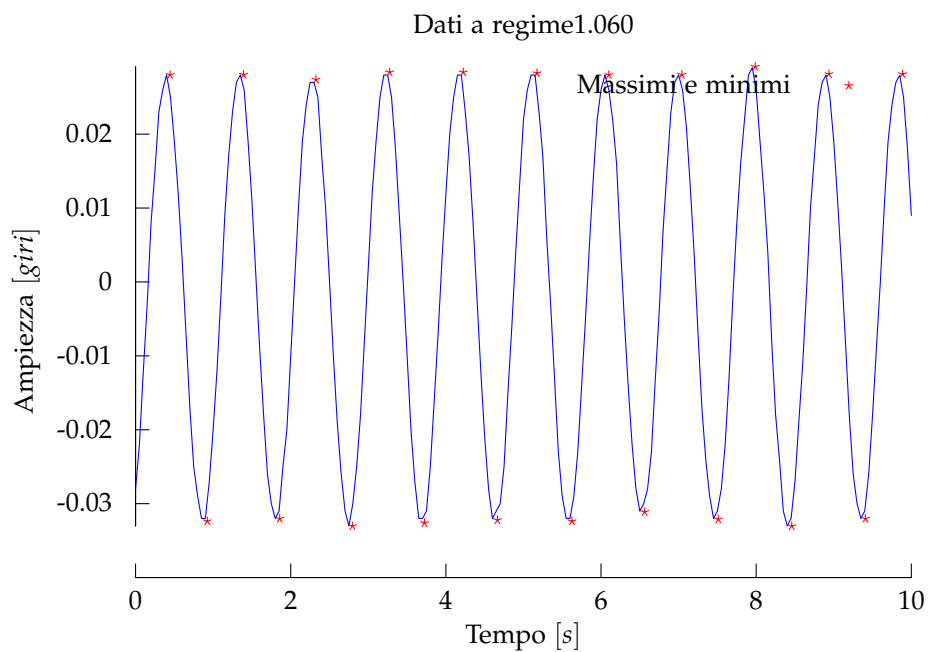


Grafico 31 Grafico 1.080dgdecad.tex

Grafico 32 Grafico 1.080greg.tex

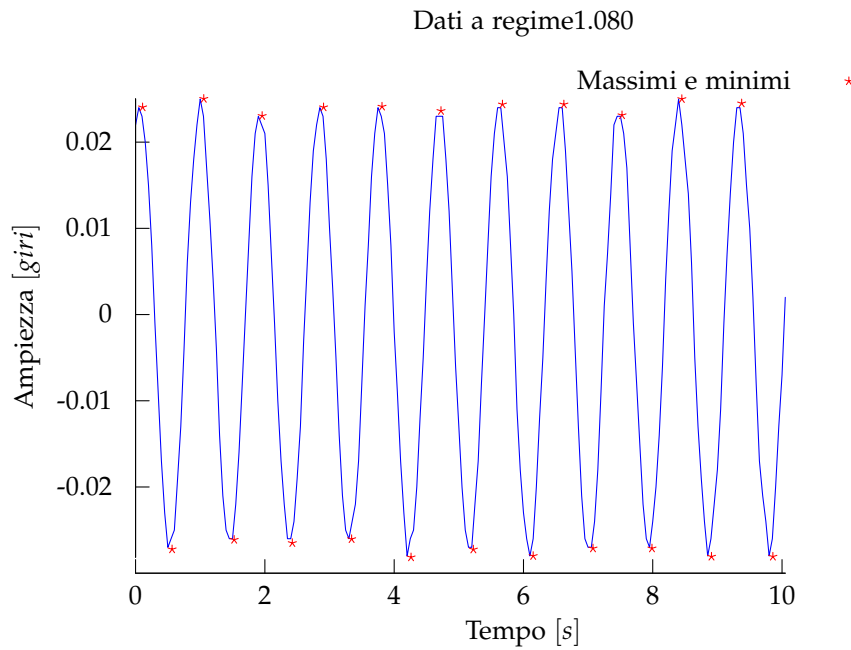
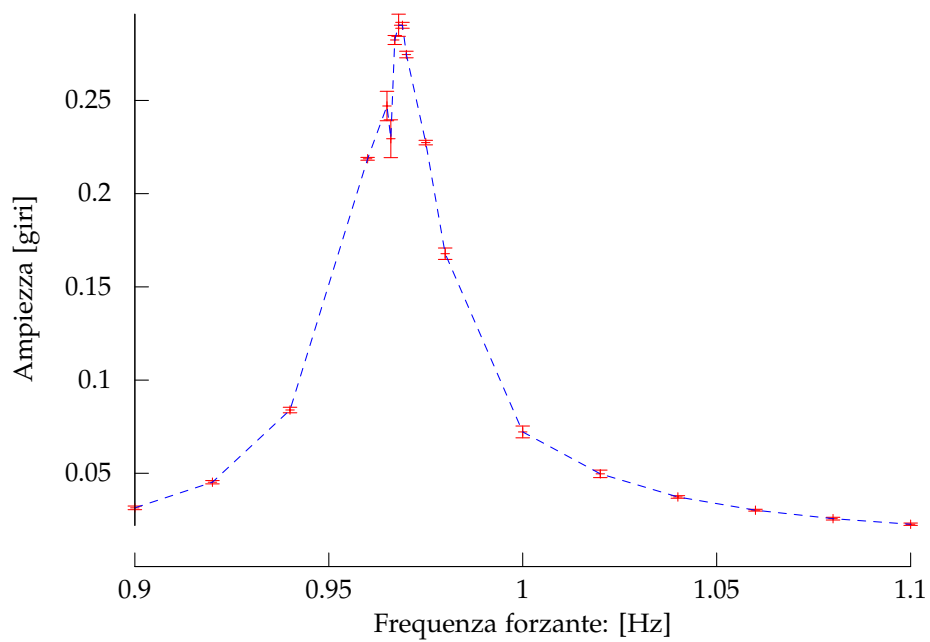


Grafico 33 Curva di risonanza



Frequenza di risonanza $(0.968 \pm 0.01)[Hz]$

IV. ANALISI DEI DATI

Come detto nella descrizione dell'apparato strumentale, il tasso di rilevamento dei dati è di 20 al secondo. Questo corrisponde a una frequenza di campionamento di 20 Hertz, di molto superiore al Nyquist rate necessario per il pendolo (il doppio della massima frequenza necessaria), dato che come verificabile a vista ha una frequenza dell'ordine di 1 Hz. Non ci sono quindi problemi di aliasing e sottocampionamento. Per quanto riguarda l'offset, è stata rifatta la calibrazione prima di ogni presa dati (inizio giornata) e si può vedere che era calibrato da un'evidente simmetria rispetto all'asse delle ascisse. Per il calcolo dei massimi è stato utilizzato un programma che riconoscesse i punti di massimo e minimo approssimando la funzione come una parabola in un intorno dei dati "stazionari" (dati massimi e minimi locali) usando il dato precedente e il successivo, vincolando la parabola a passare per questi 3 punti e trovandone il vertice. L'errore legato all'utilizzo di questa approssimazione è, come noto dallo sviluppo di Taylor delle funzioni goniometriche, $o(x^3)$, che essendo lo step 0.05 è compreso tra 10^{-3} e 10^{-4} , e trascurabile rispetto agli altri errori. Per una stima delle ampiezze legate alle frequenze di oscillazione sono stati presi i valori medi delle ordinate dei massimi (e dei valori assoluti dei minimi).

Una stima della pulsazione di risonanza è stata fatta con un processo di esplorazione iniziale che ha permesso, attraverso il metodo di bisezione, di concentrarsi sull'area nella quale l'ampiezza era più alta. Il valore finale trovato risulta di... Per stimare il coefficiente di smorzamento γ legato al movimento dell'acqua è stato tentato un approccio diretto con gnuplot, ma i problemi del suo algoritmo (Levenberg-Marquardt, una forma di step gradient descent) nel caso di funzioni come questa, in cui il gradiente dei minimi quadrati è pieno di punti stazionari locali, ne hanno impedito l'applicabilità pratica. Quindi è stato scelto un'altro approccio che elimini questi problemi, in particolare limitando lo studio a una semplice funzione esponenziale, che è stata ulteriormente semplificata in un fit lineare usando una scala logaritmica. Di conseguenza, sono stati cercati gli $x_i \mid f(x_i) = 0$. Essendo la funzione

$$\theta_0 e^{-\gamma t} \sin(\omega_s t + \phi), \quad (1)$$

poichè $e^{-\gamma t} > 0$ gli zeri della funzione sono solo gli zeri del seno. Quindi i punti medi $x_m = \frac{x_i + x_{i+1}}{2}$ fra gli zeri sono i punti in cui $\sin(\omega_s t + \phi) = 1$. Interpolando questi punti la funzione diventa dunque

$$\theta_0 e^{-\gamma t} \cdot 1 = \theta_0 e^{-\gamma t}.$$

Interpolando questa funzione con i punti $(x_m, \log f(x_m))$ (le coordinate $f(x_m)$ sono state calcolate, nei casi in cui non fossero già un punto dei dati, interpolando linearmente tra i due punti più prossimi).

La pulsazione di smorzamento è stata ottenuta attraverso una media pesata delle pulsazioni ottenute dallo studio dei periodi dei grafici durante la fase di smorzamento (vedasi tabella...). La pulsazione propria è stata trovata attraverso la formula $\omega_0 = \sqrt{\omega_s^2 + \gamma^2}$. Gli errori sono stati stimati a partire da una stima diretta con la sommatoria degli scarti al quadrato diviso $N - 1$. I grafici rivelano che, entro gli errori casuali...

V. CONCLUSIONI

VI. CODICE

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main()
{
double o0, so0, os, sos, gamma, sgamma;
cerr << "Inserire in ordine omegas, suo errore, gamma, suo errore. " << endl;
cin >> os >> sos >> gamma >> sgamma;
o0 = sqrt( ( os * os ) + ( gamma * gamma ) );
so0 = sqrt( 1 / ( (gamma * gamma ) + ( os * os ) ) * ( ( sgamma * gamma * gamma )
+ ( sos * os * os ) ) );
cout << o0 << "\t" << so0 << endl;
return 0;
}
```

%-----

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
```

```
int main()
{
    int n = 0;
    int m = 0;
    double temp;
    vector<double> data;
    vector<double> dist;
    while(cin >> temp) //Riempie il vector DATA con i dati e li conta (n)
    {
data.push_back(temp);
n++;
    }
}
```

```
sort(data.begin() , data.begin() + n);
```

```
for (int i = 0 ; i < n - 1 ; i++) //Trova i periodi
{
temp= ( data.at(i + 1) - data.at(i) ) * 2; //Fattore 2 legato al fatto che serve un periodo, cioè
dist.push_back(temp);
m++;
}
```

```
double sommasigma, somma, media, sigma = 0;
for( int i = 0 ; i < m ; i++ ) //Calcolo media
{
somma += dist.at(i);
```

```
}

media = somma / m;

for( int i = 0 ; i < m - 1; i++ ) //Calcolo sigma
{
sommasigma += ( dist.at(i) - media ) * ( dist.at(i) - media );
}
sigma = sqrt ( sommasigma / ( m - 1 ) );

double puls, spuls; //Converte tutto in frequenza [Hz]
puls = ( 2 * M_PI ) / media;
spuls = ( 2 * M_PI * sigma ) / ( media * media);

cout << puls << "\t" << spuls << endl; //Output
return 0;
}

%-----

#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
int main()
{
int n = 0;
vector <double> valori;
double temp; //Variabile temporanea contenente i valori
double temp_neg; //Variabile temporanea ausiliaria 1
double temp_pos; //Variabile temporanea ausiliaria 2
while(cin >> temp) //Mette i dati nel vector
{
valori.push_back(temp);
n++;
}
int j = 0;
vector <double> massimi; //Crea vector in cui mettere i massimi
vector <double> max_neg; //Crea vector in cui mettere elemento precedente al massimo
vector <double> max_pos; //Crea vector in cui mettere elemento successivo al massimo
vector <int> max_dist; //Crea vector in cui scrivere posizione massimi
vector <int> max_dist_neg;
vector <int> max_dist_pos;
double temp_dist; //Crea variabile temporanea per la distanza dei valori
double temp_dist_neg;
double temp_dist_pos;

for (int i = 1 ; i < ( n - 1 ) ; i++)
{
if (valori.at(i) > valori.at(i-1) && valori.at(i) >= valori.at(i+1) ) //Trova i massimi
```



```
{
temp = valori.at(i);
temp_neg = valori.at(i-1);
temp_pos = valori.at(i+1);
temp_dist = (i + 1);
temp_dist_neg = i ;
temp_dist_pos = i + 2;
massimi.push_back(temp);
max_neg.push_back(temp_neg);
max_pos.push_back(temp_pos);
max_dist.push_back(temp_dist);
max_dist_neg.push_back(temp_dist_neg);
max_dist_pos.push_back(temp_dist_pos);

j++;

}
}
vector <double> minimi;
vector <double> min_neg;
vector <double> min_pos;
vector <int> min_dist;
vector <int> min_dist_neg;
vector <int> min_dist_pos;
j = 0;
for (int i = 1 ; i < ( n - 1 ) ; i++)
{
if (valori.at(i) < valori.at(i-1) && valori.at(i) <= valori.at(i+1) )
{
temp = valori.at(i);
temp_neg = valori.at(i-1);
temp_pos = valori.at(i+1);
temp_dist = i + 1;
temp_dist_neg = i ;
temp_dist_pos = i + 2;
minimi.push_back(temp);
min_neg.push_back(temp_neg);
min_pos.push_back(temp_pos);
min_dist.push_back(temp_dist);
min_dist_neg.push_back(temp_dist_neg);
min_dist_pos.push_back(temp_dist_pos);

j++;
}
}

double delta, da, db, dc, a, b, c, vortex, vortey; //Parametri della parabola
vector <double> xverticiMAX; //Vector contenente i risultati del programma
vector <double> yverticiMAX;
```

```
double max_size = massimi.size();
double min_size = minimi.size();

for (int i = 0 ; i < max_size ; i++)
{
delta = ( ( max_dist.at(i)      * max_dist_pos.at(i) * max_dist_pos.at(i) ) +
( max_dist_neg.at(i) * max_dist.at(i)      * max_dist.at(i) )      +
( max_dist_neg.at(i) * max_dist_neg.at(i) * max_dist_pos.at(i) ) -
( max_dist_neg.at(i) * max_dist_neg.at(i) * max_dist.at(i) )      -
( max_dist.at(i)      * max_dist.at(i)      * max_dist_pos.at(i) ) -
( max_dist_neg.at(i) * max_dist_pos.at(i) * max_dist_pos.at(i) ) );

da = ( ( max_neg.at(i) * max_dist.at(i)      * max_dist_pos.at(i) * max_dist_pos.at(i) ) +
( max_pos.at(i) * max_dist_neg.at(i) * max_dist.at(i)      * max_dist.at(i) )      +
( massimi.at(i) * max_dist_neg.at(i) * max_dist_neg.at(i) * max_dist_pos.at(i) ) -
( max_pos.at(i) * max_dist_neg.at(i) * max_dist_neg.at(i) * max_dist.at(i) )      -
( max_neg.at(i) * max_dist.at(i)      * max_dist.at(i)      * max_dist_pos.at(i) ) -
( massimi.at(i) * max_dist_neg.at(i) * max_dist_pos.at(i) * max_dist_pos.at(i) ) );

db = ( ( massimi.at(i) * max_dist_pos.at(i) * max_dist_pos.at(i) ) +
( max_neg.at(i) * max_dist.at(i)      * max_dist.at(i) )      +
( max_pos.at(i) * max_dist_neg.at(i) * max_dist_neg.at(i) ) -
( massimi.at(i) * max_dist_neg.at(i) * max_dist_neg.at(i) ) -
( max_pos.at(i) * max_dist.at(i)      * max_dist.at(i) )      -
( max_neg.at(i) * max_dist_pos.at(i) * max_dist_pos.at(i) ) );

dc = ( ( max_pos.at(i) * max_dist.at(i) )      +
( massimi.at(i) * max_dist_neg.at(i) ) +
( max_neg.at(i) * max_dist_pos.at(i) ) -
( max_neg.at(i) * max_dist.at(i) )      -
( massimi.at(i) * max_dist_pos.at(i) ) -
( max_pos.at(i) * max_dist_neg.at(i) ) );

a = da / delta;
b = db / delta;
c = dc / delta;
vortex = - b / ( 2 * c );
vortey = - ( b * b - 4 * a * c ) / ( 4 * c );
xverticiMAX.push_back(vortex * 0.05); //Presente valore di conversione delle x
yverticiMAX.push_back(vortey);
}

vector <double> xverticiMIN;
vector <double> yverticiMIN;

for (int i = 0 ; i < min_size ; i++)
```

```

{
delta = ( ( min_dist.at(i)      * min_dist_pos.at(i) * min_dist_pos.at(i) ) +
( min_dist_neg.at(i) * min_dist.at(i)      * min_dist.at(i) )      +
( min_dist_neg.at(i) * min_dist_neg.at(i) * min_dist_pos.at(i) ) -
( min_dist_neg.at(i) * min_dist_neg.at(i) * min_dist.at(i) )      -
( min_dist.at(i)      * min_dist.at(i)      * min_dist_pos.at(i) ) -
( min_dist_neg.at(i) * min_dist_pos.at(i) * min_dist_pos.at(i) ) );

da = ( ( min_neg.at(i) * min_dist.at(i)      * min_dist_pos.at(i) * min_dist_pos.at(i) ) +
( min_pos.at(i) * min_dist_neg.at(i) * min_dist.at(i)      * min_dist.at(i) )      +
( minimi.at(i) * min_dist_neg.at(i) * min_dist_neg.at(i) * min_dist_pos.at(i) ) -
( min_pos.at(i) * min_dist_neg.at(i) * min_dist_neg.at(i) * min_dist.at(i) )      -
( min_neg.at(i) * min_dist.at(i)      * min_dist.at(i)      * min_dist_pos.at(i) ) -
( minimi.at(i) * min_dist_neg.at(i) * min_dist_pos.at(i) * min_dist_pos.at(i) ) );

db = ( ( minimi.at(i)      * min_dist_pos.at(i) * min_dist_pos.at(i) ) +
( min_neg.at(i)      * min_dist.at(i)      * min_dist.at(i) )      +
( min_pos.at(i)      * min_dist_neg.at(i) * min_dist_neg.at(i) ) -
( minimi.at(i)      * min_dist_neg.at(i) * min_dist_neg.at(i) ) -
( min_pos.at(i)      * min_dist.at(i)      * min_dist.at(i) )      -
( min_neg.at(i)      * min_dist_pos.at(i) * min_dist_pos.at(i) ) );

dc = ( ( min_pos.at(i) * min_dist.at(i) )      +
( minimi.at(i)      * min_dist_neg.at(i) ) +
( min_neg.at(i)      * min_dist_pos.at(i) ) -
( min_neg.at(i)      * min_dist.at(i) )      -
( minimi.at(i)      * min_dist_pos.at(i) ) -
( min_pos.at(i) * min_dist_neg.at(i) ) );

a = da / delta;
b = db / delta;
c = dc / delta;
vortex = - b / ( 2 * c );
vortey = - ( b * b - 4 * a * c ) / ( 4 * c );
xverticiMIN.push_back(vortex * 0.05);
yverticiMIN.push_back(vortey);
}

vector<int> eliminamassimi; //vector contenente la posizione dei valori da eliminare
vector<int> eliminaminimi;
int temperasemax;
int temperasemin;

for (int i = 0 ; i < max_size -1 ; i++) //Permette di trovare eventuali massimi fasulli

```

```
{
if (abs (xverticiMAX.at(i) - xverticiMAX.at(i+1)) < 0.8 ) //Intervallo considerato errore
{
if ( yverticiMAX.at(i) - yverticiMAX.at(i+1) >= 0)
{
temperasemax = i+1;
eliminamassimi.push_back(temperasemax);
} else if ( yverticiMAX.at(i) - yverticiMAX.at(i+1) < 0 && temperasemax != i)
{
temperasemax = i;
eliminamassimi.push_back(temperasemax);
}
}
}

for (int i = 0 ; i < min_size - 1; i++) //Permette di trovare eventuali minimi fasulli
{
if (abs (xverticiMIN.at(i) - xverticiMIN.at(i+1)) < 0.8 )
{
if ( yverticiMIN.at(i) - yverticiMIN.at(i+1) <= 0 && temperasemin != i )
{
temperasemin = i+1;
eliminaminimi.push_back(temperasemin);
} else if ( yverticiMIN.at(i) - yverticiMIN.at(i+1) > 0 )
{
temperasemin = i;
eliminaminimi.push_back(temperasemin);
}
}
}

int puliziamax = eliminamassimi.size();
int puliziamin = eliminaminimi.size();
int f = 0; //Variabile necessaria per regolare la modifica posizione vettori

for (int i = 0 ; i < puliziamax ; i++) //Pulisce il vector definitivo di massimi
{
xverticiMAX.erase( xverticiMAX.begin() + eliminamassimi.at(i) - f );
yverticiMAX.erase( yverticiMAX.begin() + eliminamassimi.at(i) - f );
f++;
}

f = 0;

for (int i = 0 ; i < puliziamin ; i++) //Pulisce il vector definitivo di minimi
{
xverticiMIN.erase( xverticiMIN.begin() + eliminaminimi.at(i) - f );
yverticiMIN.erase( yverticiMIN.begin() + eliminaminimi.at(i) - f );
f++;
}
```

```
}

double max_v_size = xverticiMAX.size();
double min_v_size = xverticiMIN.size();

//cout << "Le x dei massimi valgono: " << endl;
for (int i = 0 ; i < max_v_size ; i++)
{
    cout << xverticiMAX.at(i) << "\t" << yverticiMAX.at(i) << endl;
}

//cout << "Le x dei minimi valgono: " << endl;
for (int i = 0 ; i < min_v_size ; i++)
{
    cout << xverticiMIN.at(i) << "\t" << yverticiMIN.at(i) << endl;
}

/*

    int q = 0; //Algoritmo per la visualizzazione dei risultati
    cout << "Massimi: " << endl;
    for (double massimi)
    {
        cout << massimi.at(q) << endl;
        q++;
    }
    q = 0;
    cout << "A una posizione di: " << endl;
    while (max_dist.at(q) != 0)
    {
        cout << max_dist.at(q) << endl;
        q++;
    }

    q = 0;
    cout << "Minimi: " << endl;
    while (minimi.at(q) != 0)
    {
        cout << minimi.at(q) << endl;
        q++;
    }
    q = 0;
    cout << "A una posizione di: " << endl;
    while (min_dist.at(q) != 0)
    {
        cout << min_dist.at(q) << endl;
        q++;
    }
}
```

```
}
*/
return 0;

}

%-----

#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int main()
{
double temp1, temp2;
vector <double> datax;
vector <double> datay;

while(cin >> temp1)
{
cin >> temp2;
datax.push_back(temp1);
datay.push_back(temp2);
}

int n =datax.size();

for(int i = 0 ; i < n ; i++)
{
datay.at(i) = log( abs( datay.at(i) * 2 * M_PI ) );
cout << datax.at(i) << "\t" << datay.at(i) << endl;
}
return 0;
}
```

```
%-----

Media pesata
```

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <cstdlib>
#include <string>

using namespace std;

int main ()
{
    int n;
```

```
cout << "dire di quanti valor si vuole calcolare la media" << endl;
cin >> n;
double* misure = new double [n];
for (int i = 0 ; i < n ; i++)
{
    cout << "inserire valore "<< i << " della media" << endl;
    cin >> misure[i];
}

double* sigme = new double [n];
for (int i = 0 ; i < n ; i++)
{
    cout << "inserire valore "<< i << " della sigma" << endl;
    cin >> sigme[i];
}

double sommavalsig;
for (int i = 0 ; i < n ; i++)
{
    sommavalsig += (misure[i]/sigme[i]);
}

double sommak;
for (int i = 0 ; i < n ; i++)
{
    sommak += (1/sigme[i]);
}

double mediapesata;
mediapesata = (1/sommak)*(sommavalsig);

cout << " Media pesata: " << mediapesata << endl;

double errormediapesata;
errormediapesata = sqrt(1/sommak);

cout << " Error media pesata: " << errormediapesata << endl;

return 0;
}
```

Calcolo Della Gamma

```
#include <vector>
#include <dirent.h>
#include <iostream>
#include <cmath>
#include <fstream>
#include <cstdlib>
```

```
#include <string>

using namespace std;

#include <vector>
using std::vector;

class funzione_punti_lineare
{
public:
    std::vector<double> vectx;
    std::vector<double> vecty;

    //x dev'essere ordinato?
    funzione_punti_lineare(std::vector<double> vx, std::vector<double> vy)
    {
        vectx = vx;
        vecty = vy;
    }

    long double operator()(double x)
    {
        long long i = 0;
        while(vectx.at(i) < x)
            i++;

        long double x0 = vectx.at(i);
        long double y0 = vecty.at(i);

        long double x1 = vectx.at(i+1);
        long double y1 = vecty.at(i+1);

        //Coeff. angolare, DeltaY/DeltaX
        long double m = (y1 - y0) / (x1 - x0);
        long double y = m*(x-x0)+y0;

        return y;
    }
};

%-----

int main()
{
    fstream fout ("Valori.txt" , fstream::out);

    int n = 0;
```



```
double x;
double y;
vector <double> valori_x;
vector <double> valori_y; //Variabile temporanea contenente i valori
double temp_neg; //Variabile temporanea ausiliaria 1
double temp_pos = 0; //Variabile temporanea ausiliaria 2
while(cin >> x) //Mette i dati nel vector
{
    cin >> y;
    valori_x.push_back(x);
    valori_y.push_back(y);
    n++;
}

int j = 0;
vector <double> zeri; //Crea vector in cui mettere zeri
double temp_dist; //Crea variabile temporanea per la distanza dei valori
double temp_dist_neg;
double temp_dist_pos;
double temp_dist_no = 0;
double temp_dist_forse;
double temp_dist_si;

for (int i = 0 ; i < ( n - 1 ) ; i++)
{
    if (valori_y.at(i) == 0)
    {
        temp_neg = abs((valori_x.at(i) + valori_x.at(i + 1))/2) ;
        temp_dist = abs((temp_neg + temp_pos)/2);
        temp_pos = temp_neg;

        zeri.push_back(temp_dist);
        i++;
    }
    else if (valori_y.at(i) * valori_y.at(i+1) < 0 ) //Trova gli zeri
    {

        temp_neg = abs((valori_x.at(i) + valori_x.at(i + 1))/2) ;
        temp_dist = abs((temp_neg + temp_pos)/2);
        temp_pos = temp_neg;

        zeri.push_back(temp_dist);

        if (zeri.size() > 1 )
        {
```

```
        cout << temp_dist << endl;
    }

    j++;

}

}

funzione_punti_lineare f(valori_x, valori_y);

for (double xzero : zeri)
{
    fout << xzero << "\t" << f(xzero) << endl;
}

return 0;
}
```